Table of contents

- Introduction;
  - License;
  - Existence of the documentation;
  - Author Notes;
- ShiVa Engine;
  - Quick history;
  - Proprietary formats;
  - Libraries and Tools;
    - Obtaining a copy of ShiVa Engine;
- Datamining;
  - Pre-requirements;
    - Setting up Termux;
  - Tools;
  - Obtaining and extracting the archives;
  - Structure of a ShiVa Engine game;
    - More Proprietary formats;
  - Using the tools;
    - PVRTexToolCLI;
    - Unluac;
      - Batch converting with Bash;
    - CompressonatorCLI;
    - NinjaRipper;
    - Noesis;
      - Importing plugins;
      - Exploring and exporting meshes;
- Debugging;
  - On Android;
  - On Linux;
  - On Windows;
- Modding;
  - Creating Unofficial ports;
  - Changing the Game Assets;
    - Editing Audio;
    - Editing Textures;
      - Converting one into DDS;
    - Editing Scripts;
  - Repacking and Testing;
  - Messing with ShiVa;
    - Disabling Animations;
    - Swapping Meshes;
      - First Method;
      - Second Method;
  - Distributing your Mod;
- Epilogue;
- Sources;

Introduction

Welcome to the **ShiVaDM** (**D**ata**M**ine) documentation!
This is your host, Baempaieo, offering a bunch of information that should document exhaustively *everything* pertaining to datamining, debugging, and modifiying every single game made with *ShiVa Engine*!

The following should make the reader come out with all of the important knowledge that there is to know regarding the topic (after read all of it, that is):
but before venturing into such, there are some **very** important points that should be aknowledged first

License

All of this documentation is licensed under `CC BY-NC 4.0` (Creative Commons Attribution-NonCommercial), which means you can remix, redistribute, and create derivative work off of it:
with the condition of not using the material for commercial purposes and **must** give credit where is due.

I'd like to keep all of this knowledge **free** for everybody, available for the whole internet to share and learn from.
If you want to read further details regarding the license, feel free to give a look towards the `LICENSE` file.

Existence of the documentation

It all started around 2019, as far as i can remember, and a much younger me was trying to get a model extracted from a game made in said game engine.

Since i *really* didn't knew a lot of what was i doing (datamining, file extensions, file headers, etc.) in-depth, all i've used to do was, at max, fiddling with said unknown file on a hex-editor on Android.
Honestly, reason why i wanted to do so was purely as a *self-achievement*, all without disclosing all on the details as to how:
all until, one day, i had saw a random guy (which i'll keep it's username a secret) over Instagram.

Apparently they found out how to do the same thing i wanted to do, plus including pictures of a random model opened into a 3D manipulation program....but there was a **very evident** catch: **clout**. Lots of it.
They left the process "a secret" on purpose, so that a person ignorant on the matter could have asked "how" in the comments, which then led them to get bait-and-switched into paying for a script via the author's PayPal.

I really wasn't liking the whole thing, so, many days later i've got in touch with the user through Discord (since they were the first to discover such a thing).
If i were to nutshell the whole situation (and add a veil of comedy), it all boiled down to me asking and get the following as a response:

> *"Just figure things out and use IDA Pro lmao"*

and that's probably where, to be really frank, it was the biggest "nothing burger" advice i've ever got in my life- that is, force myself to use proprietary software to figure out an undocumented file format. And had to pretend that the file in question was a compiled executable (even though my younger self never knew IDA Pro's use-case back in the day).

After aknowledged that, i had went on an years-long journey on figuring out the latter and the remaining proprietary file formats produced by the game engine.
It was a constant on-and-off adventure, some times giving up (and thinking all of this was futile), other times relapse all of my past discoveries to learn something new about it, gave up, then (hopelessly) came back to it 1-2 years later after gathered more knowledge on tech-
but long story short, it has took me **5 years** to figure all of this stuff.

As of late April 2024, all of this coming-and-going stubborningly has led me to understand *fully* what was actually going on:
almost every single nook and cranny of ShiVa Engine, all by using free and open-source software (even though it all boiled down to `hexedit`, `wine`, `cabextract`, and an handful of other stuff).

Sure, i had found out that since 2019 i was going after the *wrong file*, but i am glad to announce that since 2023 (which in that year i've got a major uplift towards my tech knowledge) and 2024 (which i've gathered far more than the latter):
i've gathered all of that new knowledge and decided to **share it to the rest of the world!**

Author Notes

- The information included into this document is compatible with every asset produced under any version of the engine (`2.0.0`, `1.9.2`, and older versions);
- Some information may be subject to changes between a commit and another, as such is updated to contain fully-informed, researched, and tested notions about the game engine- this getting rid of inaccuracies and/or wrong assumptions about some of it's components (and more);
- Throughout the sections, i would be using a lot of `UNIX-like` terminal commands here and there. This is due to me using *Linux* as my daily-driver;
- The game that will be analyzed throughout this documentation (as a way to offer practical examples, too) will be Dust: Offroad Racing, a racing game made in ShiVa by *CATMOON Productions*;
    - This game would be *slightly* deviated in some portions, even though it all boils down to file formats that never got introduced in this very game at all;

- As of `19-03-2026`, **ShiVaDM** will stop being distributed into different formats (like `.epub`, `.pdf`, `.txt`) for agevolating easier mantainability of the repository. Because of this choice, the docs will be exclusively mantained as a `markdown` document;
- Take note that, while one ShiVa game will be analyzed throughout this whole documentation, **ShiVaDM** also counts as an universal guide for every game made in *ShiVa Engine* itself;
- Take *also* note that **ShiVaDM** starts and ends with a streamlined process. If you skip one process, you won't get the next one without proper context;
    - Keep in mind that *all* of the portions are interconnected (from *ShiVa Engine* to *Modding*);

ShiVa Engine

Moving on, here we are, straight to the game engine itself: **ShiVa Engine**.
Into this section, it'll be introduced stuff like:
it's history, the proprietary file formats it brings along, libraries and tools (that it uses for producing it's proprietary assets), and how to obtain a copy of the engine nowadays!

Quick history

ShiVa Engine was a cross-platform 3D, closed-source, game development suite made by *ShiVa Technologies SaS* (formerly called *StoneTrip*, a company based in France).
In terms of the game engine itself, it was originally conceived as **Shiva Suite** back in 2004, which then became **StoneTrip Development Kit** (or **STDK**) back in 2006 (alongside a brand new "web player" named **STON3D Web Player**) and established the logo that we all know and love ever since.
Back in 2007, the whole game-development suite got reverted into **ShiVa**, which included a suite of other programs packed-in (such as `Ston3D`, `Ston3D Server`, `StonBox`, and `StonStudio`).
As time went on, the ShiVa branding has got multiple polishes and iterations and established it's name **back in 2010** as **ShiVa3D**, all the way up to **2015**.

As of 2013, the company has went under liquidation[archive.org]. According to the interview, that was just a way for the company to start fresh than anything.
Because of the latter, back in 2016 *StoneTrip* has been rebranded[archive.org] into *ShiVa Technologies SaS* (even though the change has took effect since 2013, several months after the liquidation), and their landing page redirected to **shivaengine.com** the same year.

Ever since **2015**, ShiVa has published version `1.9.2` of their game engine, but they didn't do much than a small GLSL update back in 2021.
Sadly version `2.0` has never materialized from the official website, despite ShiVa has done several blog posts about it (such as a 2012-13 Roadmap, Beta 2 Release, Beta 3 Release, Beta 4 Release, Beta 5 Release, Beta 6 Release, Beta 7 Release, Beta 8 Release, and a Beta 9 Release. Curiously enough, some of these blog posts do mention that ShiVa used Qt4-5 as it's underlying GUI toolkit).
As of early 2025, though, someone has archived that exact version on the Internet Archive (albeit being just an unregistered copy).

As more time went on, the engine's status has been quietly dying. Fast forward to March 2023:
the company has announced ShiVa Engine's EOS. Ever since that, ShiVa (the company) has decided to leave untouched platforms like their Facebook, YouTube (got wiped as of 2025, but you can see a backup of it here), and GitHub (official announcement blog post).

Ever since the game engine reached it's **E**nd **O**f **S**upport, someone reached out to their Facebook page, questioning if it'll ever become open-source someday.
As of 2023, this is how the opportunity got greeted as

> *"Because there are still customers/companies relying on ShiVa being closed source. If this ever changed, we would certainly think about it."*
>
> *- ShiVa Technologies' response, coming from their latest Facebook post* [archive.org]

Meaning, in other words, that they would look into it *if* any customer/company would give up it's usage for other alternatives (and stop being over-reliant to the engine due to it's closed-source status)

Proprietary formats

And since the engine's nature is closed-source: it won't be enough to introduce it's own proprietary file formats, too!
Of course, this is just an appetizer for the whole *datamining* section of this documentation.

As of `10-01-2025`, after tinkered enough with all of the 3 archive formats, these *might* be just one type of archive with minor changes here and there (since all of them share the same file structure, hierarchy, and all use the `zlib` compression algorithm under the hood)

File Extension	MIME Type	Header (Hex)	Header (Unicode)	Description

---

File Extension	MIME Type	Header (Hex)	Header (Unicode)	Description

`.stk`	*ShiVa3D STK Document*	`06 08`
		`00 00`
		`00 20`
		`20 20`
		`20 20`
		`20 20`
		`20 xx`
			`.....`	*A proprietary archive format. As seen from this screenshot, this file format is referred as a `Runtime Package`. These can be used either as a full game project or a supplementary DLC/s to an already-existing project. Every file with this extension can be named to everything the developer wishes.*

`.ste`	*ShiVa3D STE Document*	`04 08`
		`00 00`
		`00 20`
		`20 20`
		`20 20`
		`20 20`
		`20 xx`
			`.....`	*A proprietary archive format. According to the image linked previously, such is referred as an `Edition Archive`. There isn't much proof about it, but it might have been the former de-facto standard Runtime Package before `.stk` has came out as a substitute.*

`.smf`	*ShiVa3D SMF Document*	`05 08`
		`00 00`
		`00 20`
		`20 20`
		`20 20`
		`20 20`
		`20 xx`
			`.....`	*A proprietary archive format. Files with this extension can only be named as `S3DMain.smf`, and can be exclusively found into ShiVa-powered games built for mobile devices (like Android). Reason why such files are named as such is due to a library that the Authoring tool bundles into the compiled project itself: `S3DClient.so` (which is strictly related to `ShiVa3DClient`), and it's only task is to read and play files that are named `S3DMain.smf` stored into the application package itself (like an `.apk` or `.ipa` file). If the `S3DMain` package gets either replaced or renamed in a way or another, the game will refuse to boot up. Besides this, `.smf` files are actually glorified `.stk` projects, since the Authoring Tool (the game compiler that the suite comes packaged with) renames the inputted `.stk` project into `S3DMain.smf` (plus changes it's file signature) just to work with the `S3DClient.so` library*

Libraries and Tools

Moving on, if you ever wish to install ShiVa on your machine (whether it'll be under Windows or running it under **Wine**) and want to explore it's internals:
it surprisingly exposes almost every tool and library that it uses (both for it's proprietary formats and what runs under the hood).

The following can be found at

```
C:\Program Files(x86)\ShiVa Technologies\ShiVa Editor Web\Tools
```

And if you were to list all of the contents from that folder, you'll get exposed to these peculiar tools & libraries.
For simplicity's sake, i had commented out everything i could to get a proper pointer *which* tool or library does *what* within the game engine (even though i had to research and test the tools firsthand to figure their purpose)

> ***Author's Note:***
> *The full folder path and date are excluded from the following output*

```
Luac.exe              ---> Short for 'Lua Compiler', this tool compiles any .lua script into
.lub. Runs version 5.0.3, 32-bit variant;
PVRTexTool.exe        ---> Short for 'PowerVR Texture Tool'. It's a tool made by Imagination
Technologies that transcodes multiple image formats into .tga, .tex, .dds, and .pvr textures.
This is the tool responsible for creating textures;
ShiVaTextureTool.exe  ---> An in-house tool made by ShiVa. It is unsure if this is a GUI for
PVRTexTool;
WMAEncode.exe         ---> Windows Media Audio encoder;
cudart32_30_14.dll    ---> An external library related to Nvidia's CUDA technology. This is the
very first lib that ties with NVidia's tooling;
ffmpeg2theora.exe     ---> A fork ffmpeg that transcodes multiple media formats either into Ogg
Theora (.ogv, video) or Ogg Vorbis (.ogg, audio);
jpeg62.dll            ---> JPEG Image Library;
lame.exe              ---> LAME executable for encoding different audio formats into .mp3 (even
this one a FOSS program!)
libeay32.dll          ---> Another library, this one's related to OpenSSL (Open Secure Socket
Layer);
libpng12.dll          --.
```

```
                        > Image format libraries: these being respectively PNG and TIFF
encodings;
libtiff3.dll          --'
lua2cpp.exe           ---> This tool converts Lua code into C++;
msvcp100.dll          --.
msvcp71.dll             |
msvcp80.dll             -> Microsoft Visual C++ Redistributable libraries.
msvcr100.dll            |  These libraries are the only clue that the engine has been coded in
C++
msvcr71.dll             |  and compiled in Visual Studio
msvcr80.dll           --'
nvcompress.exe        --.
nvdecompress.exe        > Nvidia-related tools. The first 2 are related for texture compression
& decompression.
nvdxt.exe               |  'nvdxt.exe' converts textures, and 'nvtt.dll' is the library used for
the above tools;
nvtt.dll              --'
oggdec.exe            --.
                        > Ogg Vorbis decoder and encoder;
oggenc2.exe           --'
openssl.cfg           --.
openssl.exe             > OpenSSL tools. These are, in order: a config file, it's executable,
and it's library;
ssleay32.dll          --'
zlib1.dll             ---> This is the compression library that the engine uses for creating
it's own .stk, .smf, and .ste archives!
```

Obtaining a copy of ShiVa Engine

Since the engine and everything else went kaput in 2023, people nowadays cannot obtain the engine from any official sources:
yet, luckily, thanks to the existence of webarchive, we can grab any copy of it via snapshots in the present!

In terms of platforms that got salvaged throughout the years, **Windows** is the one that has (almost) everything:
from various versions of the game engine (`1.6.2`, `1.7.0`, `1.9.2`, and `2.0.0`), with the addition of `ShiVa Authoring Tool` (which i'll explain it later on in the documentation) and `ShiVa Web Player` (a small extension for your OS that can play website-compiled ShiVa games on your browser).
As of MacOS, very little has been salvaged than an older version of `ShiVa Authoring Tool` for iPhone devices, a newer stand-alone version of the latter, and it's own port of `ShiVa Web Player`.
Last but not least, Linux has got the shorter end of the stick, this by having *just* the `ShiVa Web Player` extension saved:
even though there's nothing to worry, as Linux (and every other `UNIX-like` OS) can execute the Windows installation package for the engine through Wine.

Besides all of these salvaged platforms, let's move back the focus to the *game engine* itself.
The only version that the average user can use is the `WEB` edition:
yet, this edition is only usable for exploring the contents of an extracted (or ready to be exported as an `.stk`/`.ste` archive) project-
which makes it an impromper candidate to do anything else with.

As of `04-08-2026`, i had found out a tool (named ShiVa Advanced Edition 1.9.0+ Patch) that could crack the `ADVANCED` edition of ShiVa:
but upon my discovery, i had found out that doing anything with the real deal feels clunky and impractical.
To add more to the latter, every single asset you'll import will get converted into it's proprietary equivalent, and you must jump into lots of hoops to edit any of them in the first place-
but despite that, these roadblocks can be easily bypassable by *ditching* ShiVa itself and use a mixed assortment of FOSS (one proprietary) (which will be covered later on in the documentation) and internal tools (that the engine uses under the hood) to make (almost) anything possible!

> ***Author's Note:***
> *The following table will only display links for ShiVa Engine and related tools for playing `.stk` files.*
> *If you want see more than that, feel free to check my own ShiVa Engine Archive.*

Platform	stonetrip.com	shivaengine.com	shiva-engine.com	softpedia.com	archive.org
Windows	ShiVa 1.6.2 PLE	`N/A`	ShiVa 1.9.2.0 WEB - ShiVa 1.9.2.0 BASIC - ShiVa 1.9.2.0 ADVANCED needs activation key - ShiVa Authoring Tool 1.4.0.0 - S3D Web Player 1.8.1.0	ShiVa PLE 1.7.0.1	ShiVa Editor 2.0.0 Unregistered Copy

---

Platform	stonetrip.com	shivaengine.com	shiva-engine.com	softpedia.com	archive.org
MacOS	S3D Authoring Tool for iPhone 1.6	S3DMain Player	ShiVa Authoring Tool 1.4.0.0 - S3D Web Player 1.8.1.0	N/A	N/A
Linux	N/A	N/A	S3D Web Player 1.8.1.0	N/A	N/A

Datamining

And here we are, the juicier part of the documentation itself: ***datamining***!
From this point onwards, the rabbit hole will go deeper over this topic, and so it will be exploring the data within a ShiVa-powered game-
but before that, some preparation is needed firsthand:

Pre-requirements

As anything that is related to datamining, we must know a bunch of things before we'll put our hands on the tools that will be introduced later on.

To start everything in order, this whole thing can be done with **every OS** (as long as it is an `UNIX-like` or an `NT-based` system. This also includes *Android* aswell, even though you must need Termux installed first. Will introduce a setup guide on it shortly after).
With that out of the way, here are the pre-required tools (& knowledge) we must need before heading over to the tools we'll use for datamining:

- **A terminal emulator**, this is *mainly* important for Linux and Android, but such will be partially used on Windows- depending on *which* tool;
- **OpenJDK**, this is a ***must have*** tool that should be installed firsthand on our machine. For *vanilla builds*, hop over OpenJDK.org, find the **jdk.java.net/** hyperlink, download the archive for your machine, and follow the install instructions here. If you're on Linux (or Termux), you can get the tool by using the distro's package manager's search feature (`apt search jre` or `apt search openjdk`) and install it. For *Microsoft builds*, you can refer to Microsoft's OpenJDK Builds and get the installer for your OS and architecture;
- **Basic terminal knowledge**. There's nothing scary about it, this requirement explains everything on it's own;
- *Lots, lots of patience!*

Setting up Termux

> ***Author's Note:***
> *The following tutorial assumes that you already have installed Termux-X11 on your Android device.*
> *If you're following the following on a device that runs Android 14+, you must install the `nightly` build of the app (else it won't work).*

After finding out that **termux-box** has been discontinued a long time ago, this section will introduce, instead, how to install **Hangover**!
To start everything up, let's go into termux and type the following

```
termux-setup-storage        # Sets up storage permission for the app
apt update; apt upgrade     # Updates the termux repos
```

If you're having trouble with the commands above, write this one to fix the problem

```
termux-change-repo
```

and with that, we have Termux ready to roll!

To install Hangover, just write this command

```
apt install tur-repo hangover
```

With the whole setup done, using Hangover is pretty simple:
just write the following in the terminal (replace `[program name]` with the program's name)

```
hangover-wine [program name]
```

If you want to run graphical programs through Hangover:
we must get the `x11-repo` repository, X11, a window manager, and a terminal emulator to go alongside it

```
apt install x11-repo termux-x11 fluxbox aterm
```

If you have the *nightly* build of `Termux-X11` installed, replace `termux-x11` with `termux-x11-nightly`.

After gotten the needed packages, we can start an X11 session with this command

```
termux-x11 :0 -xstartup "fluxbox"
```

and if you open the `Termux-X11` app, you should see a window manager running!

To open the terminal emulator, just do a right-click, then select `aterm`:
and from that you can reuse the `hangover-wine` program to open *any* GUI program.

To close the session, just come back to the Termux app, then do the `Ctrl-C` key-combo.

After done everything, you can now run programs like `QuickBMS` at your fingertips!
If you'd like to know more about Hangover, feel free to give a look to their Github repository.

Tools

Moving on, here we are in the *actual* tools section of this documentation.
The following tools listed in here were the ones i had found in my years-long journey in order to unpack/extract *almost* every asset from any ShiVa-powered game.

- **QuickBMS** GitHub mirror, archive.org
  - A specialized tool for unpacking and repacking peculiar archive formats from a vast selection of games and programs. This will be used for unpacking our fellow `.stk`, `.ste`, and `.smf` archives that were documented earlier;
    - Since we'll look upon how to extract the latter, make sure to get the shiva.bms script firsthand;
  - If you're using Linux as your daily driver and would rather use a GUI than a terminal, feel free to give a look towards **AmorBMS**. Setup should be easy and it's usage is perfectly compatible with this documentation;
- **PVRTexToolCLI**
  - This tool should help up with converting (and decompressing) every `.pvr`, `.dds`, and `.tex` texture into `.png` and/or `.jpg`. I'd vividly suggest to only get the archive that contains *the Windows executables* inside, since all of the tarballs made for Linux distros contain dependencies for old and deprecated versions of numerous programs and libraries;
- **Compressonator** archive.org
  - This tool will be used to convert our `.atc` (actually `.ktx`) textures straight into `.png`, but can also be converted into `.jpg`, `.dds`, and more. Make sure to head over to the **Releases** page, grab the `compressonatorcli-4.5.52-win64.zip` file, and decompress it firsthand. For convenience's sake we'll use just the *Windows* edition of the CLI tool;
- **Unluac** archive.org
  - This tool will be used for decompiling into `.lua` every single `.lub` script from the game. This program requires a *Java Runtime* installed on your machine, since the tool was written in Java. If you haven't installed `OpenJDK` yet on your machine, i'd suggest going back to the **pre-requirements** section, then come back to this portion once done;
- **NinjaRipper** *Windows program. Needs Wine installed on Linux/MacOS*
  - This tool will be used to rip all of the meshes we'll see in the game. Unlike **3D Model Researcher**, this tool will make mesh-ripping a breeze- yet both are incapable to extract bone data entirely. Make sure to download the `ninjaripper171.7z` file and decompress it firsthand;
- **Noesis** archive.org *Windows program. Needs Wine installed on Linux/MacOS*
  - This tool will be used in conjuction with **NinjaRipper**, which will allow us to import the `.rip` files and export it's contents into a readable model format. Make sure you had downloaded the `.zip` file and extracted it's contents in a folder (which we'll call it `Noesis` for convenience's sake);

The rest of the tools below are just an optional, but, depending on the game or scenario you're into:
these can either be useful for faster research or if you want to *just view* the mesh models.

- **Basic Linux Utils**. This extends to using tools like `grep`, `less`, `more`, pipes (`|`), and *some* basic knowledge of **bash scripting**;
- **i3DConverter**;
  - An universal, closed-source and cross-platform, 3D viewer and converter for numerous file formats. Only con is that, according to this spreadsheet archive.org, any mesh (or `.msh`) file coming from ShiVa Engine can only be loaded and viewed, but not converted and saved at all. To make things worse, it can only import models coming from **Dust: Offroad Racing**;

Obtaining and extracting the archives

> ***Author's Note:***
> *This section assumes that you've already prepared (and unpacked) QuickBMS and downloaded the game's Android package inside the `Download` folder of your machine, but also assumes that you already have an archive manager installed out-of-the-box.*
>
> *Before proceeding, make sure you've both moved inside the `Download` folder (either with the terminal or your file manager of choice) and gathered the needed tools by consulting the **tools** section.*

As mentioned from the proprietary formats section:
ShiVa Engine introduces over 3 new proprietary archive formats, and into libraries and tools have also discovered the compression library it refers to for creating them.

Alright, with that out of the way, it's time to cover on *how to obtain* such from an `.apk` file, how to use **QuickBMS**, and document along the way the extra proprietary formats it will introduce post-extraction.

Starting off, i have downloaded the `.apk` file for the game we'll be inspecting upon (again, **Dust: Offroad Racing**), which i had conveniently renamed as `Dust.apk` for the sake of this guide.
Moving on, let's move inside the `Downloads` folder of our machine (this is where typically our downloaded files are at):
since i use a terminal for my workflow, i simply move inside a folder by doing

```
cd Downloads
```

else, if you're using a graphical file manager (like *Explorer*, for instance), you can simply double-click on the folder.

Going along, it is now our time to check inside the downloaded game!
To check what's inside, we must need an *archive manager*, since `.apk` files are actually a `.zip` archive.
Whether the archive manager works graphically or not, what matters is that you can use it without issues yourself.
Since i use the CLI version of `7z`, this is how i tend to check the contents of an archive

```
7z l Dust.apk
```

and upon the output of the command, this is what we should (hierarchically) find inside the game

```
   Date      Time    Attr         Size   Compressed  Name
------------------- ----- ------------ ------------  ------------------------
2012-09-18 18:49:20 .....         1629          698  META-INF/MANIFEST.MF
2012-09-18 18:49:20 .....         1750          828  META-INF/MYKEY.SF
2012-09-18 18:49:20 .....         1383         1112  META-INF/MYKEY.RSA
2012-09-10 19:12:38 .....     44315768     44315768  assets/S3DMain.smf
2012-09-18 18:03:58 .....         6258         6258  res/drawable/app_icon.png
2012-07-18 14:46:04 .....       129138       129138  res/drawable/app_splash.png
2012-09-18 18:10:12 .....          564          291  res/layout/billing_not_supported.xml
2012-09-18 18:10:12 .....          640          322  res/layout/main.xml
2012-09-18 18:10:12 .....         5244         1521  AndroidManifest.xml
2012-09-18 18:10:12 .....         2428         2428  resources.arsc
2012-09-18 18:03:58 .....        12818        12818  res/drawable-hdpi/app_icon.png
2012-09-18 18:03:58 .....         6258         6258  res/drawable-ldpi/app_icon.png
2012-09-18 18:03:58 .....        12818        12818  res/drawable-mdpi/app_icon.png
2012-09-18 18:03:58 .....        12818        12818  res/drawable-xhdpi/app_icon.png
2012-09-18 18:10:10 .....       277188       121015  classes.dex
2012-09-18 18:09:54 .....       916168       463984  lib/armeabi/libcrypto.so
2012-09-18 18:09:52 .....       170875        75109  lib/armeabi/libopenal.so
2012-09-18 18:09:38 .....      8697384      2764151  lib/armeabi/libS3DClient.so
2012-09-18 18:09:52 .....       204072       107287  lib/armeabi/libssl.so
2012-09-18 18:09:54 .....       891568       456160  lib/armeabi-v7a/libcrypto.so
2012-09-18 18:09:54 .....       151511        69239  lib/armeabi-v7a/libopenal.so
2012-09-18 18:09:52 .....      8250484      2611400  lib/armeabi-v7a/libS3DClient.so
2012-09-18 18:09:54 .....       197308       104331  lib/armeabi-v7a/libssl.so
------------------- ----- ------------ ------------  ------------------------
2012-09-18 18:49:20           64266072     51275752  23 files
```

As we can see here, the file we need to extract (`S3DMain.smf`) can be found inside the `/assets/` folder- great!

To note more, aside the `.smf` file, we can also verify that a game is made in ShiVa even by inspecting further what there is inside the `/lib/armeabi*/` folder aswell- but why am i mentioning this?
Remember when we have looked upon the **libraries and tools** section from much earlier?
Alright, whenever the game engine packages the *whole* game for Android, several libraries get actually pulled inside the `/lib/armeabi*/` folder path. Each library exactly corresponds to a library used into the engine itself, like:

- `libS3DClient.so` is the one responsible for reading and executing the `S3DMain.smf` file (inside the `/assets/` folder);

- `libssl.so` establishes secure socket connections when communicating to a server;
  - `libcrypto.so` is the library that, altogether with `libssl.so`, adds encryption whenever your device and the game's servers communicate with eachother (so that no external attacker would get in and compromise the online traffic between the 2);
- `libopenal.so` adds playback for the `.ogg` audio data contained within the `S3DMain.smf` archive (which the `openanal` in `libopenal` stands for **Open A**udio **L**ibrary...and which audio encoding is free and open-source? *Ogg Vorbis*!);

Moving on, it is now time to extract the `.smf` archive from the game.
Again, since i use a terminal, this is how i tend to extract one specific content from any archive

```
7z x Dust.apk assets/S3DMain.smf
```

and after extracted the latter, not only it would be outputted inside a folder named `assets`, but it also our time to shine with **QuickBMS**!

If you had prepared the tool already, i'd suggest creating 2 folders in particular:
one for containing all of the extracted files from the `quickbms.zip` archive (and move the `shiva.bms` script inside), another where we will output the contents of the `S3DMain.smf` file at.
Since, again, i use the terminal, i'll create the following

```
mkdir ~/Downloads/quickbms      # this'll be the folder for, obviously, QuickBMS
mkdir ~/Downloads/test          # meanwhile this'll be the folder for containing the .smf's
extracted contents
```

Now, time to use the tool- but **halt**, using QuickBMS actually differs between *Windows and Linux*:
but both have in common the habit of moving firsthand to the folder where the executable is at- which, in this case, it should be

```
cd ./quickbms
```

and then execute the tool like this

```
quickbms -G     # this is for Windows. It executes the tool in GUI mode

chmod +x ./quickbms     # this is for Linux. This command makes the tool executable
./quickbms ~/Downloads/quickbms/shiva.bms ~/Downloads/assets/S3DMain.smf ~/Downloads/test/    #
and this extracts the contents of the .smf archive
```

If you're following this section with **AmorBMS**, you can skip the above commands and:

- Type `~/Downloads/quickbms/shiva.bms` in the 1st text field;
- Type `~/Downloads/S3DMain.smf` on the 2nd text field;
- And type `~/Downloads/test` on the last text field;

Once done the above, click on the `Extract` button

When the tool is done extracting the contents from the archive, you can either close the window or press any key to close the tool:
but aside that, we have officially learned on how to find, extract, and unpack `S3DMain.smf`!

After all of this, though...what about `.stk` archives?
`.stk` archives are quite the hard find to get around, but depending on what you're looking after, you can find them up into this very folder path (after installed the `WEB` edition ShiVa Engine on your machine, that is)

```
C:\Program Files (x86)\ShiVa Technologies\ShiVa Editor Web\
```

even though the process boils down to the one i had described earlier, it can serve as a great way to practice on how to use QuickBMS better (if this guided tutorial wasn't enough).

What about `.ste` archives?
You would be in real surprise how easy they are to obtain, even after ShiVa has died for some time already.
If you give a deep dive towards *ShiVa's official Github page* (which you can find a link to it back to the quick history section) you may find a great handful of repositories that does contain a *lot* of those archives.

Structure of a ShiVa Engine game

After successfully learned together on how to extract a `.smf` archive, we can now finally look upon the general structure of a ShiVa Engine game as a whole.

By cutting things short, the output below will have comments that explain, atleast to a fundamental degree, which file is responsible for what it does inside the game itself (and, as stated previously, explain, describe, and document each and every proprietary format after this very section)

```
Games
|_______  Dust.gam                  # This folder portion contains the "core" that ShiVa reads
to start the game;

Models
|_______  Model_Core_Camera_Back.mdo # This one contains model data
          ...

Resources                           # Any ShiVa game has this particular folder.
|                                   # This one nests the rest of the assets, which, in-engine,
get called externally by the files
|                                   # contained within 'Games', 'Models', and 'Scenes'
|_______  AIModels
|           |_______  AI_Core_Main_CarConfigurator.aim       # Not to be confused with *actual*
game models- this one sets the behavior of a specific thing
|                     ...                                   # if an action happens in-game.
|_______  Fonts
|           |_______  DefaultFont.fnt                       # A simple folder containing fonts,
but aren't .otf or .ttf files in disguise
|                     ...
|_______  HUD
|           |_______  HUD_Core_Main_CarConfig.hud           # HUD data
|                     ...
|_______  Materials
|           |_____    Collider.mat                          # Material textures
|                     ...
|_______  Meshes
|           |_____    Car_1_lod1_Polygonal_Geometry_lib.msh  # Mesh data
|                     ...
|_______  Musics
|           |_____    Music_Core_InGame_1.mus               # Music data
|                     ...
|_______  Particles
|           |_____    Particle_CheckpointGate.par           # Particle textures
|                     ...
|_______  Scripts
|           |_____    AI_Core_Main_CarConfigurator_Function_GetAIVar.lub     # This folder
contains the compiled .lua scripts
|                     ...
|_______  SoundBanks
|           |_____    Soundbank_Core_Cars.snb               # SoundBanks collect all of the
needed sounds from the 'Sounds' folder,
|                     ...                                   # which get assigned to various game
elements (ex. characters, actions, etc.)
|_______  Sounds
|           |_____    Sound_Core_ButtonClick.snd            # Sound data
|                     ...
|_______  Textures
|           |_____    DefaultFont.dds                       # Texture data
|                     ...
|_______  Trails
            |_____    Trail_Core_WheelBurn.tra              # Trail textures

Scenes
|_______  Scene_Core_Dummy.scn                              # Scene data. This one gives
instructions how animations, models, and shaders
          ...                                               # must be put to compose an in-game
scene
```

After given a pretty good look at how this ShiVa game is structured, we would now proceed moving over to the next title:
as it will uncover what these files are *really* about and what they actually contain.

More Proprietary formats

As mentioned briefly in the previous title, this section will purely document the technical info about (more) of it's proprietary formats **Dust: Offroad Racing** after extracted it's own `.smf` file.

In this table, not only there would be documented all of the file formats that we've looked at in the previous title:
but also other file formats that come included in any other *ShiVa-powered* game aswell.

File Extension	Header (Hex)	Header (Unicode)	Folder Path	Description
.gam	04 00 00 00 4E 63 50 xx	....NcP	/Games/	*Plain text data. These files are a list that calls all of the files in the assets, which then get used to compose the game itself. Each and every asset is called in a specific order*
.mdo	03 00 00 00 4E 63 50 xx	....NcP	/Models/	*Plain text data. These files are a list that searches for every mesh, sub-mesh, texture, animations, and skeletons to compose things like characters, maps, and more. If one or more files listed in an `.mdo` file can't be found in the assets, the engine would either: not render (applies to meshes), render but the mesh spazzes (applies to skeletons), draws but the mesh is black (applies to textures), T-pose (applies to animations), or silence (applies to audio) the missing asset*
.scn	03 00 00 00 4E 63 49 xx	....NcI	/Scenes/	*Plain text data. These files work like `.mdo` does, but it's used for composing game scenes*
.aim	03 00 00 00 4E 63 50 xx	....NcP	/Resources/AIModels/	*Plain text data. These files call a specific set of AI Model-related scripts from the `/Resources/Scripts/` folder. In ShiVa Engine, CPUs get called as "AI Models"*
.anb	03 00 00 00 4E 63 49 xx	....NcI	/Resources/AnimBanks/	*Plain text data. These files work like it's namesake. These list animations contained within `/Resources/AnimClips/`, which then compose a "bank" to be used exclusively by a character, object, etc.*

File Extension	Header (Hex)	Header (Unicode)	Folder Path	Description
.ani	03 00 00 00 4E 63 58 xx	....NcX	/Resources/AnimClips/	*Raw animation data. These files references a mesh's corresponding skeleton from `/Resources/Skeletons/`, then follows which bone moves first and last*
.fnt	03 00 00 00 4E 63 58 xx	...NcX	/Resources/Fonts/	*Plain text data. These dummy files references a font's spritesheet contained within `/Resources/Textures/`*
.hud	03 00 00 00 4E 63 50 xx	....NcP	/Resources/HUD/	*Plain text data. These files call internally components like sound, scripts, and textures to compose an HUD*
.mat	03 00 00 00 4E 63 58 xx	....NcX	/Resources/Materials/	*Plain text data. These files references textures from `/Resources/Textures/`, then renders them as materials on a model*
.msh	03 00 00 00 4E 63 50 xx	....NcP	/Resources/Meshes/	*Raw mesh data. These files contain a mesh's UV map, vertices, and face data*
.mus	4F 67 67 53	OggS	/Resources/Musics/	*Audio data. These are simply `.ogg` files with it's extension renamed*
.par	03 00 00 00 4E 63 58 xx	....NcX	/Resources/Particles/	*Plain text data. Works like `.mat` does, but the textures are referenced to compose particles*
.pmp	03 00 00 00 4E 63 58 xx	....NcX	/Resources/PixelMaps/	*Plain text data. These are dummy files with nothing inside*
.rmp	03 00 00 00 4E 63 49 xx	....NcI	/Resources/RenderMaps/	same as .pmp
.lub	1B 4C 75 61 50 xx xx xx	.LuaP	/Resources/Scripts/	*Lua script. This is compiled with the (`32-bit` variant) `Luac` utility. These can be easily decompiled with external tools. According to it's official manpage, such compilation is done both for enhancing performance and avoid eventual tamperings of a script*

File Extension	Header (Hex)	Header (Unicode)	Folder Path	Description
.ske	03 00 00 00 4E 63 58 xx	....NcX	/Resources/Skeletons/	*Raw skeleton data. These files contain matching raw skeleton data that can be put together with it's corresponding mesh from* */Resources/Meshes/*
.snb	03 00 00 00 4E 63 58 xx	....NcX	/Resources/SoundBanks/	*Plain text data. Works like* *.snb* *does, except it nabs the soundfiles from* */Resources/Sounds/*
.snd	4F 67 67 53	OggS	/Resources/Sounds/	Same as .mus
.dds	44 44 53 20	DDS	/Resources/Textures/	*Texture data. This format is natively supported on Windows due to being Microsoft's own proprietary image format*
.pvr	34 00 00 00 00 02 00 00 00 02 00 00 09 00 00 00 19 83 01 00 00 AB 02 00 04 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00 50 56 52 21	4.....PVR!	/Resources/Textures/	*Texture data. This format is Imagination's proprietary "PowerVR" texture data. You can check out it's file format specification document for more technical details*

File Extension	Header (Hex)	Header (Unicode)	Folder Path	Description
.etc	34 00 00 00 00 02 00 00 00 02 00 00 09 00 00 00 36 01 01 00 B8 AA 02 00 04 00 00 00 FF FF FF FF FF FF FF FF FF FF FF FF 00 00 00 00 50 56 52 21	4.............6.....................PVR!	/Resources/Textures/	*Texture data. These are just `.pvr` files with it's extension changed*
.atc	AB 4B 54 58 20 31 31 BB 0D 0A 1A 0A 01 02 03 04 00 00 00 00	.KTX 11.....	/Resources/Textures	*Texture data. These are `.ktx` textures with their extension renamed to `.atc`. KTX is short for "Khronos TeXture", an open-source texture format made by Khronos Group, and this type of texture is specifically the `KTX v1` variant. You can check out it's file format specification document for more technical details*
.xml	3C 3F 78 6D 6C 20 76 65 72 73 69 6F 6E 3D 22 31 2E 30 22 20 65 6E 63 6F 64 69 6E 67 3D 22 55 54 46 2D 38 22 20 3F 3E	<?xml version="1.0" encoding="UTF-8" ?>	/Resources/XML/	*XML document. In ShiVa, these can be highly versatile for tons of situations: from text strings to external plugins*
.tra	03 00 00 00 4E 63 58 xx	....NcX	/Resources/Trails/	*Plain text data. Works like `.mat` and `.par` does, but the textures are referenced to compose trails*

Using the tools

Since 2 sections ago we have looked on how to extract the files from `S3DMain.smf`, we will now come to the point of *converting* a great chunk of these proprietary assets into usable and readable data- and this is where the tools i've listed earlier come into play!

Keep in mind that only 3 of them require the *usage of the terminal* (this meaning **unluac**, **PVRTexToolCLI**, and **CompressonatorCLI**), while the rest is purely a GUI program (in the case of **NinjaRipper** and **Noesis**).
If you're intentioned to just convert the textures or the `.lua` scripts, you can do the following (for convenience's sake)

- Move the **unluac** binary into `/Resources/Scripts/`;
- Move the **PVRTexToolCLI** executable into `/Resources/Textures/`;
- Move the `/Resources/Textures/` folder into the `compressonatorcli-4.5.52-win64` folder;

Once done the latter, it's now time to rumble!

PVRTexToolCLI

> ***Author's Note:***
> *Before converting the `.etc` textures, make sure to rename their file extension into `.pvr`*

Once moved into the `/Resources/Textures/` folder path, open the terminal and write the following command

```
PVRTexToolCLI -noout -ics sRGB -i Tex_Core_CheckpointGate_Star.pvr -d
Tex_Core_CheckpointGate_Star.png
```

and you should be good to go!

If you'd like to know more complicated usages of the tool, feel free to give a quick read at Imagination's official documentation[archive.org] to get a deeper understanding on the utility.

Unluac

Once moved into the `/Resources/Scripts/` folder path, open the terminal and write the following command

```
java -jar unluac AI_Shell_Main_Init_Function_HUD_Garage_ShowBuyCar.lub -o
AI_Shell_Main_Init_Function_HUD_Garage_ShowBuyCar.lua
```

Batch converting with Bash

As a last thing, here's a very useful trick i tend to do often in Bash myself.
Sure, it's just a `for` loop, but it makes the job far easier when you want to mass-convert a collection of files:
to get the point across, here's some boilerplate template below

```
for file in *.[input file extension]; do [tool] [input flag] "\$file" [output flag] "\$file".
[output file extension]; done
```

Though, if you want to see some extra examples to digest the technique better:
here's more, but with actual CLI tools used in the mix

```
for file in *.{pvr,dds}; do PVRTexToolCLI -noout -ics sRGB -i "\$file" -d "\$file".png; done

for file in *.lub; do java -jar unluac "\$file" -o "\$file".lua; done
```

If you're following this portion with a Windows machine (as of `20-12-2024`), *Batch* isn't as flexible as Bash is in terms of `for` loops: so, as a way to recover from this lack of flexibility, i'd heavily suggest to get Chocolatey on your machine and install Git afterwards. The following should contain a special feature named *GIT Bash*, which you can find it once opened your file explorer and opened the context menu.

CompressonatorCLI

> ***Author's Note:***
> *Before converting the `.atc` textures, make sure to rename their file extension into `.ktx`.*
> *Also, make sure to replace the `[filename]` placeholder into the file's actual name.*

After moved into the `compressonatorcli-4.5.52-win64` folder, do the following steps

- Create a folder where to output the converted textures. Let's call it `Output`;
- Open the terminal, then you can pick between the 2 commands below;

If you want to convert a singular texture, you can write this

```
compressonatorcli.exe -ff KTX -fx PNG Resources/Textures/[filename].ktx Output/[filename].png
```

Otherwise, if you want to batch convert multiple textures, you can write this instead

```
compressonatorcli.exe -ff KTX -fx PNG Resources/Textures/ Output/
```

and everything should be good to go!

NinjaRipper

> ***Author's Note:***
> *Before proceeding, make sure you had learned on how to obtain and extract an archive, modify it's assets (specifically converting every texture into DDS), and repack the game for `Windows`.*
> *If you don't have the time to do either, you can grab my own unofficial PC port of the game, follow the "How to install" instructions for `Windows`, then come back to the documentation afterwards.*
>
> *This is the only optimal way to get the meshes extracted from any ShiVa-powered game.*
> *If you hook up NinjaRipper on any generic "Android emulator" for Windows (if said game was made for Android):*
> *not only it will require extra steps to follow, but the device must be capable of virtualization aswell- a thing not every device has.*

After extracted the archive, do the following steps:

- Get into the `ninjaripper1.7.1` folder;
- Get into the `x86` folder;
- Double-click on the `NinjaRipper.exe` file;

If you have Wine installed, open a terminal, then write the following command

```
wine Downloads/ninjaripper1.7.1/x86/NinjaRipper.exe
```

If all went well, we should be greeted by **NinjaRipper**'s GUI!
Now, to setup the program to rip the game's meshes, do the following steps:

- Go on the `Target (DX 11/9/8/7/6 application)` group;
- On the `Exe:` field, click on the `...` button on the right;
- A window should appear afterwards. We must find a program called `S3DEngine_Windows.exe`;
- After found it, double-click on it's name;
- Move to the `Arg:` field and write `-f=0 dust.smf`;
  - If you're ripping another game, you can replace `dust.smf` with the game's respective `.smf` file;
- Move down to the `Output Directory` group;
- On the `Dir:` field, click on the `...` button on the right;
- After the window has opened, go to your `Downloads` folder and create a subfolder named `Ninja`;
- Once created, highlight it, then click on the `Ok` button;
- With everything done, click on the `Run` button placed on the top right;

If the game has started successfully, we'll be greeted by **Dust: Off-Road Racing** in windowed mode!
To finish everything off in beauty, follow these last steps:

- Click on the `PLAY` button;
- After loaded, we'll be greeted by our starter vehicle;
- If you want to rip it's model, just press the `f10` key on your keyboard;
- If the game freezes for a couple of seconds, this means the tool has started to dump what is being displayed in-game;
- After the game has resumed from it's freezing state, close it;
  - If you want to dump more meshes: keep the game open, find more models, and repeat the previous 3 steps until you're satisfied;
- Close NinjaRipper;

To check if the models have ripped successfully:

- Head over to the `Ninja` folder, then `_NinjaRipper`;
- If you'll see a folder finishing with the name `_S3DEngine_Windows.exe_`, get into it and it's subfolder;
- Lots of files ending with `.rip` and `.dds` should show up;

If you don't see anything, this might mean either the tool hasn't ripped anything or you've picked the wrong folder:
even though this is easily solvable by reopening NinjaRipper and follow back the steps after launched the game.

Alas, we have finally learned on how to dump meshes *straight from a ShiVa-powered game!*

Noesis

After we had done the latter with **NinjaRipper**:
we can now use **Noesis** to export the models from the `.rip` files the tool has dumped earlier.

Though, before heading into the tool, we must learn the following things:

- Importing plugins;
- Exploring and exporting meshes;

Importing plugins

This process would be pretty easy, but: why do we have to learn this in the first place?

Well, the answer is pretty simple: **Noesis** doesn't have native support for `.rip` files-
so, patch up this lack of support, we must learn on how to import plugins in the first place.
Not only this'll be a "set it and forget it" procedure, but it will also extend the tool's functionalities aswell.

To get things done quickly, follow these steps:

- Get into the `ninjaripper1.7.1` folder;
- Get into the `tools` folder;
- At last, get into the `noesis_importer` folder;
- Select both files and copy them;
- Get out of the `ninjaripper1.7.1/tools/noesis_importer/` folder path;
- Get into the `Noesis` folder;
- Go to the `plugins` folder, then paste the files in it;
- Drag the `fmt_ninjaripper_rip.py` file into the `python` folder;
- Unpack the `ninjaripper_noesis_modified_by_blackninja.zip` archive;

And we're good to go!
Now, let's move on to the next section.

Exploring and exporting meshes

With the latter done, we can now open **Noesis** for exploring and exporting the contents of the dumped `.rip` files!

To open the tool, just go to the `Noesis` folder and double-click on the `Noesis.exe` file.
Upon launching it's executable, we'll be greeted by **Noesis**' window.

If you have Wine installed, open a terminal, then write the following command

```
wine Downloads/Noesis/Noesis.exe
```

With the above done, let's get straight into business and follow these steps:

- Move to the GUI's left side. From there, you will see a tree-view of your filesystem;
- Search for the `Ninja` folder;
- Once found, double-click it;
- Double-click on the `_NinjaRipper` folder;
- Double-click on the folder that ends up with `_S3DEngine_Windows.exe_`;
- Lastly, double-click on it's subfolder;

If everything went well, we should see a list of files appearing in the middle part of the UI.
If we were to click on any of the `.rip` files, we should see stuff appear on the right side of the UI: like models or textures!

If you had found the mesh that you want to export (for example `Mesh_0006.rip`), do the following steps:

- Right-click on it;
- Click on `Export`;
- A window called `Export Media` should pop-up;
- If you want to change the model's format, there's a drop-down menu after `Main output file:`;
- Let's pick `.obj - WaveFront OBJ` from the menu;
- If you want to change the export path for the model, click on the `Browse` button after `Destination file(s):`;
- Let's pick the `Downloads` folder;
- Let's call our exported model `car` (or any other name you could think of);
- Click on `Save`;

- Click on the `Export` button;
- Wait for the model to export. A pop-up should appear once that's done;
- Close the window;

You can repeat the process as many times as you like, as long as **Noesis** isn't closed.

After exported the mesh, you can merge it with it's own texture to gain a complete model:
which is easily doable with Blender- just make sure to flip the texture vertically beforehand, since ShiVa stores textures upside-down by default.

Yet, this is it: we have finally learned on how to *rip and export* models from a ShiVa-powered game!

Debugging

> ***Author's Note:***
> *This section assumes that you either have a Linux or Windows machine with ADB and a terminal emulator installed:*
> *or, depending on how you're following this section, an Android device with a root manager and a terminal emulator app installed.*

And here we go, another valuable piece to our documentation: ***debugging***!
In this section, we'll explore on how to debug our ShiVa-powered game via *Android*, *Linux*, and *Windows*:
of course, just like most of this documentation, this'll require heavy usage of the terminal (which would be partly used on *Windows*).

Since i had never saw any piece of official documentation that discloses such process, i will share this valuable undocumented knowledge right into my documentation:
yet, if you're debugging a ShiVa-powered game on *Android* or *Windows*, you *must* boot it once to generate it's own logfile (`S3DMain.log`).

Regardless of platforms, though, keep in mind that the logs will always overwrite the previous one each and every time the game is launched.

On Android

If you're eager to debug a ShiVa game on Android, it is surprisingly doable to some capacity.

Since the game's logfile is kept inside `/data/data/[appID]/app_data/`, which is at the device's root folder:
you *must* have root permissions in order to get the `S3DMain.log` file in the first place.

To start everything in order, if you have a PC:
your Android device *must* be already hooked via an USB connection, with **Developer Options** and **USB Debugging** enabled.
If you don't know how to do such, give a read on this official guide[archive.org].

With the latter done, simply start the ADB deamon on the terminal

```
adb start
```

And to gain root priviledges with it, write this other command

```
adb root
```

Otherwise, if you don't have a PC but use a root manager (**ex.** `Magisk`, `KernelSU`) on your device:
write this command to gain root priviledges, instead

```
su
```

Afterwards, just follow these (commented) commands in your terminal

```
cd /data/data/com.catmoonproductions.dust/      # Let's get inside Dust's app folder
cd app_data/                                    # Then it's own 'app_data' folder
ls -l                                           # As last, list it's contents
```

With the last commands done, your terminal should output the following:

```
total 28
drwx------ 3 u0_a573 u0_a573  4096 2026-08-17 13:06 4ee121e9
```

```
drwx------ 2 u0_a573 u0_a573  4096 2026-08-17 13:06 Cache
-rw------- 1 u0_a573 u0_a573 13357 2026-08-17 13:06 S3DClient.log
drwx------ 2 u0_a573 u0_a573  4096 2026-08-17 13:06 Saves
```

And here it is, we had found the logfile!

To copy it over on your device and read it with a text editor, just type this command

```
cp S3DMain.log /sdcard/
```

The above should copy the `.log` file at the root of your internal storage.

With that done, you can rinse and repeat the process as many times as you like.
It all consists of opening the app, get it's `.log` file, then analyze it to troubleshoot what happened on the recent session.

If the whole tutorial hasn't satisfied you enough, you can always rely on using the `logcat` command.
This command outputs *everything* that happens on your device, but also what the game does under the hood.
You can either invoke it like this and see the logs outputted on the terminal

```
su -c logcat
```

Or redirect it's output on a plain-text file, like this

```
su -c logcat >> /sdcard/file.txt
```

On Linux

Unlike Android, Linux is slightly more involved in comparison:
yet it doesn't require any root priviledges in the first place, but it involves more the usage of one of the internal tools included within **ShiVa Authoring Tool** than the whole thing-
which i'll introduce it after setted up the latter.

In order to install the previously mentioned tool, we must move all the way back to [obtaining a copy of ShiVa Engine] from several sections ago.
What we really need is just the `Authoring Tool` from the *Windows* row.

Once downloaded said executable, we must get into our `Downloads` folder

```
cd Downloads
```

and then install the *Authoring Tool* with this command, this by using Wine

```
wine shiva_authoring_tool_1.4.0.0.exe
```

and with that, we should be at the installation wizard. Just follow along with it, since it's pretty easy to go through.

Now though, you may be asking, *what* is the tool that we need for debugging our picked game on Linux?
the answer may sound silly, but the **Authoring Tool** has inside a tool named `S3DEngine` (short for `ShiVa3D Engine`).
The following is capable of reading and executing `.ste`, `.stk`, and `.smf` packages, but beyond that, not only we can debug our favorite ShiVa-powered game, but we can also execute it with custom arguments aswell, such as:

- A custom "environment" XML document hosted on a server (`-c=[PATH]`/`--server=[PATH]`);
- Custom window height and width (`-w=NUM`/`--width=NUM` and `-h=NUM`/`--height=NUM`);
- Fullscreen mode (`-f=0-1`/`--fullscreen=0-1`. `0` forces windowed mode, while `1` enables fullscreen (default option));
- Failsafe mode (`-F=0-1`/`--forcefailsafe=0-1`);
- Custom anti-aliasing (`a=0-3`/`--appaalevel=0-3`. `0` sets the antialias as defined into the engine's config file, `1` disables antialiasing, `2` sets antialiasing to `2x`, while `3` sets it to `4x`);
- In-Game Shadow levels (`-s=0-1`/`--appshadowlevel=0-1`);
- Post-rendering effects (`-p=0-1`/`--apppostrelevel=0-1`);
- Reflective surfaces (`-r=0-1`/`--appreflectlevel=0-1`);

With the arguments already introduced, let's get straight into the meat-and-potatoes of this portion and show how to use the tool.
To find `S3DEngine`, we must go to the following folder path

```
cd .wine/drive_c/'Program Files (x86)'/'ShiVa Technologies'/'ShiVa Authoring
Tool'/Data/Windows/Linux/Build
```

and, since we'll be presented by many variants (`headless`, `32-bit`, and `64-bit`) of it, just write in your terminal

```
chmod +x ./S3DEngine_Linux_x86_64.bin
./S3DEngine_Linux_x86_64.bin --help
```

and you'll be welcomed by an `help` menu on how to use the tool.

Now, where could we find in-handy an `.smf` or `.stk` file to test out the waters?
Of course, we can refer to our previous attempt on how to obtain and extract one and run it with the tool we've found out right now!

Since our `.smf` file is placed inside `~/Downloads/assets/`, we can give it to the tool like this

```
./S3DEngine_Linux_x86_64.bin ~/Downloads/assets/S3DMain.smf
```

and upon executing this command, we'll be greeted not only with the game executing on our PC:
but also a wonderful verbose output into the terminal, so that we can thoroughly debug for any issues, errors, or performance with the whole game *in real time*!

On Windows

> ***Author's Note:***
> *Make sure to replace `[username]` with your own user account's name*

Meanwhile on Windows:
it's more of a mix between Linux's way of doing things, but with Android's way of storing the `S3DMain.log` file.
Setting up **ShiVa Authoring Tool** is pretty much identical to Linux, except you don't need both `wine` and a terminal.

I'll spare the setup instructions, since you just need to pick `ShiVa Authoring Tool 1.4.0.0` from the table at obtaining a copy of ShiVa Engine.
The setup should be pretty straightforward even in here.

Just like on Linux, even Windows has it's own `S3DEngine` program to playback `.stk` files, which can be found at

```
C:\Program Files (x86)\ShiVa Technologies\ShiVa Authoring
Tool\Data\Windows\Windows\Build\S3DEngine_Windows.exe
```

After we had found out the program:
open the command line, then get into the folder by writing the following command

```
cd C:\Program Files (x86)\ShiVa Technologies\ShiVa Authoring Tool\Data\Windows\Windows\Build\
```

In terms of usage and it's arguments, they're just the same as on Linux.
To invoke it, just write the following in the terminal

```
S3DEngine_Windows.exe --help
```

and just like on Linux, we'll be greeted by an huge wall of text scrolling down on the screen.

And to launch an `.stk`/`.smf` file with Windows' version of `S3DEngine`, just write the following command:
which happens to work just the same way as Linux's

```
S3DEngine_Windows.exe C:\users\[username]\Downloads\assets\S3DMain.smf
```

And lo and behold, now the game's running on Windows!
This method can *also* be used to run natively the game on PC, besides being used just for troubleshooting issues with the game.

Unlike Linux, though, the Windows port is incapable of printing any debug text via the terminal window. Why's that?
Apparently `ShiVa Technologies SAS` has compiled the tool *without* verbose output, while the Linux edition has the opposite.

Good for us that, whenever a ShiVa-powered game gets launched under Windows, `S3DEngine` will generate a peculiar folder in `AppData/Roaming/`.
To find it, just do the following steps:

- Open the Start Menu;
- Type in `%AppData%` into the search field;
- After typed the `Send` button, Explorer will open up;
- Get into `Roaming`;

After done the steps above, we'll find a folder named `Stonetrip`!
Get into it, then get into the `Player` folder and....here we are, we had found `S3DMain.log`!

From this point onwards, just open the file with any text editor of your choice (like `Notepad` or `WordPad`):
and lots of debug text will show up. Just like on Linux, even this one is completely usable on troubleshooting issues with the game.

Want to do the same even on `MacOS`?
Feel free to check back the [obtaining a copy of ShiVa Engine] section and download the `S3DMain Player` zip file.
The following should be under the **shivaengine.com** column, in the *MacOS* row.
Never had any experience with MacOS before, so i'll leave the whole installation of the tool as a pure guessing game.

With that out of the way, *happy debugging!*

Modding

Last but not least, here we are on the last section of this whole documentation: ***modding***!

If you may have followed all of the sections before this one, i had actually teased a way on how to modify (or *mod*) a game made with ShiVa, but also previously introduced a tool named **ShiVa Authoring Tool** aswell.
Now, here's the good news, this particular program can actually allow us to compile our `.stk`/`.smf` files into a functioning game, meaning that we can *create unofficial ports of our selected ShiVa-powered game!*

To start things off, though, we may need the tools for doing all of this:
which it all boils down to having at-hand the tools [we had introduced into the "Datamining" section] (even though i'd suggest just keeping **QuickBMS**, **Unluac**, **PVRTexToolCLI**, and your favorite text editor), and the ones that were introduced [while we were looking together at the engine's internal tools and libraries] (even though this just boils down to **Luac**, **FFmpeg**, **PVRTexToolCLI**, and **CompressonatorCLI** (for `.ktx` textures)).

All of these tools are a *must* to have in order to:
convert, edit, convert back, and re-importing our edited assets (plus passing the edited package to the *Authoring Tool* to compile it).

Creating Unofficial ports

> ***Author's Note:***
> *This method won't work with some games.*
> *Some developers like to design their own game to work exclusively with a particular CPU architecture and platform:*
> *but despite that, people can still use the Authoring Tool and edit the game as much they like- you just need to pick the same target platform and architecture that the devs picked while compiling the project.*
>
> *For simplicity's sake, i'll show the process for Windows only:*
> *since, more or less, compiling for other platforms boils down to the same process.*
>
> *If this portion ain't enough and want to learn on how to compile games for other platforms (`Android`, `PalmOS`, etc.), you can check [this official piece of documentation] for a step-by-step tutorial (some of these platforms, like `Android`, may require either a Windows 7 VM or machine in order to work).*

Since generally porting a game to another platform is, typically, pretty difficult and time consuming:
this is where ShiVa's *cross-platform* nature starts to shine, allowing every type of person to port their favorite game with extreme ease!

The very first thing we must do, if we're intentioned to feed to the tool an `.smf` file to compile up, we must do these 2 commands firsthand

```
cd ~/Downloads/assets/      # This is where our previously extracted 'S3DMain.smf' file is at
mv S3DMain.smf Dust.stk     # And we'll be renaming it into 'Dust.stk', both for ease to find it
& for the Authoring tool to compile it without issues
```

and when done so, we must create a copy of the file into an USB (must be `8GB` or higher in storage size):
so that, when we pick it from the file explorer, it'll be easier and quicker to find.

After that, time to open **ShiVa Authoring Tool**.
If you're using Linux, you must type the following in the terminal

```
wine .wine/drive_c/'Program Files (x86)'/'ShiVa Technologies'/'ShiVa Authoring Tool'/'ShiVa
Authoring Tool.exe'
```

and the tool will open in a snap!

To give a very quick guide on the UI:

- On the left you'll find a vast selection of platforms (like `Windows`, `Linux`, etc.) to port the game into;
- Meanwhile, on the right, there are 2 big buttons going on;
    - The one on the left guides you on how to use the tool, meanwhile the one on the right gets straight into the compilation process;

Since we want to port our game to `Windows`, we must do the following:

- Select **Windows** on the menu on the left side of the UI;
- Move to the right side of the UI and then select `Or get started now...`;
- Once there, we must select the `.stk` file we must compile from. To find it, we must hover on the `Application pack` field, and then click on the purple folder (placed in the right corner of the text field). When clicked, it should pop-up a file selection menu;
    - If you're on Windows, select the drive letter where your USB is mounted at, then pick the `Dust.stk` package by double-clicking it;
    - If you're on Linux, you must select the `Computer` icon, then select the `E:\` drive, and pick the `Dust.stk` package by double-clicking it;
- Once picked, click on the `Build` button (placed on the bottom right corner of the UI) and let the Authoring tool do it's magic;

When the whole procedure is done, you now have on your hands **Dust: Offroad Racing** unofficially ported for *Windows*!
The port should be saved in your USB, which is exactly the place you've imported the file from, and will be outputted as `Dust_Windows.zip` by the Authoring tool.

To play the game, you need to extract the `.zip` archive, of course.
Again, in my case i'll do it like this

```
cd /media/[username]/USB            # Here i'll get inside the USB the project has been exported
to
7z x Dust_Windows.zip -oWindows     # And here i'll extract the archive, and place the files in
a folder named 'Windows'
```

as a last thing, all there misses is executing the `.exe` file to play it!

Of course, on `Windows` is all a matter of getting into the folder, and then double-click on the `Dust.exe` file:
while on `Linux` it'll be

```
cd Windows/
wine ./Dust.exe
```

and as a final extra, if you were to ever write alongside the executable the `--help` argument, you'll find out that the Authoring tool packs a standalone version of the `ShiVa3D Engine` executable (which is yanked from `/Windows/Build/` and renamed):
only difference is that it's configured to play the `.stk` file packaged inside (by default), but you can still use it to execute other `.stk`, `.ste`, and `.smf` packages into your machine aswell.

And with this, *this is how you can unofficially port a ShiVa-powered game to any platform you like!*

Changing the Game Assets

> ***Author's Note:***
> *This portion can be interchangibly done either before or after you had reimported the game assets into it's dedicated `.stk`/`.smf` file.*
>
> *If you want to go for the easier road and see any instant results from modifying the assets:*
> *feel free to jump on the repacking and testing portion, import the original assets as-is, then come back to this portion for editing the assets post-reimport.*
>
> *While this order-switching applies to almost every asset in a ShiVa game: textures are completely exempt from this rule.*
> *You must convert all of the non-DDS textures into DDS (which will be shown as a sub-portion in the **Editing Textures** portion), delete all of the non-DDS textures, then reimport the textures in order to display them proper in-game.*

This section is moreso dedicated as an introduction on changing/editing game assets from a ShiVa-powered game.
In order to edit an asset from a ShiVa project, we should know on how to extract `.stk`, `.smf`, and `.ste` files by using **QuickBMS**
(if you don't know yet how, feel free to look back on how to obtain and extract them, then come back here once learned).

Beyond that, in the next portions there would be practical examples on how to:

- Edit and replace Audio assets;
- Decompress, edit, and compress back textures;
- Decompile, edit, and recompile a `.lub` bytecode script;

Other filetypes are automatically excluded, since the rest can be either boiled down to:

- Being proprietary to **ShiVa**, but can be easily swapped or replaced by renaming the new file into the one you're replacing;
    - This only works if both files share the same extension and came from a *ShiVa-powered* game (`.msh` with `.msh`, `.ske` with `.ske`, etc.);

- Showing (almost) the same process that will be demonstrated later on;
- Being dummy files with plain-text data in it. If these get either swapped or edited, the engine will ignore it;
    - Surprisingly enough, `.mdo` files are exempt from this;

Editing Audio

To edit an audio asset, we must get inside the following folder path and list it's contents

```
cd ~/Downloads/out/Resources/Sound/
ls -l
```

and from here we'll pick whatever soundfile we want to edit from- let's say we want to pick `Sound_Core_ButtonClick.snd` and replace it with a nicer sound effect, for example.

If you want to edit the file, do the following commands on your terminal

```
cp Sound_Core_ButtonClick.snd ~/Downloads/    # Let's create a copy of the file in the Downloads folder
cd ~/Downloads                                # Move into it
mv Sound_Core_ButtonClick.snd Sound_Core_ButtonClick.ogg    # And change it's extension to the real one
```

Otherwise, if you want to replace the file with a better one, this is where we'll use `ffmpeg` to convert it-
but before conversion, though, we must know the original properties of the original file, so that the engine will handle it without issue.
This is where `ffprobe` will come in handy for us

```
ffprobe Sound_Core_ButtonClick.ogg
```

once we get an output from it, the audio file should have `libvorbis` as it's audio codec.

Once checked the latter, we'll now proceed on replacing it.
For example purposes, the file we'll convert and use as a replacement will be called `ButtonNew.mp3`

```
ffmpeg -i ButtonNew.mp3 -acodec libvorbis Sound_Core_ButtonClick.ogg    # acodec's short for 'audio codec'
```

if the tool may ask to replace the audio file in the process, type `y` in the terminal to proceed.
When everything's done, just do the following in the terminal

```
rm ButtonNew.mp3    # Let's remove the original copy of the new soundfile
```

And, in the end, for both methods, we must follow the commands below

```
mv Sound_Core_ButtonClick.ogg Sound_Core_ButtonClick.snd            # Rename the file's extension back to '.snd'
mv -f Sound_Core_ButtonClick.snd ~/Downloads/out/Resources/Sounds/   # Replace the old SFX with the new one
```

Editing Textures

> ***Author's Note:***
> *Make sure to replace the `[filename]` placeholder with the file's actual name!*

Meanwhile, to edit a Texture asset, we must move inside

```
cd ~/Downloads/out/Resources/Textures
ls -l
```

and, again, pick the file we want to edit- let's say we want to pick `DefaultFont.dds` and give it a different color with our image manipulation program.

Before editing the file, we must do the following commands

```
cp DefaultFont.dds ~/Downloads    # Let's create a copy of the original in the Downloads folder
cd ~/Downloads                    # Then move inside of it
PVRTexToolCLI -noout -ics sRGB -i DefaultFont.dds -d DefaultFont.png   # Decompress the file
with PVRTexToolCLI
rm DefaultFont.dds                # And remove the original '.dds' file, just to make sure
```

and once edited the texture, we must do the following to convert and replace the texture

```
cd ~/Downloads/compressonatorcli-4.5.52-win64/
compressonatorcli.exe -ff PNG -fx DDS ../DefaultFont.png ../DefaultFont.dds
cd ..
rm DefaultFont.png
mv -f DefaultFont.dds ~/Downloads/out/Resources/Textures/
```

Otherwise, if you'd like to edit a `.ktx` texture, we will use **CompressonatorCLI**.
Let's go back again to our textures folder, like this

```
cd ~/Downloads/out/Resources/Textures/
```

and then do these commands

```
mv [filename].atc [filename].ktx    # Replace the texture's extension into '.ktx'
cd ~/Downloads/compressonatorcli-4.5.52-win64/   # Get into the tool's folder
```

Then, do these of commands to convert the texture

```
compressonatorcli.exe -ff KTX -fx PNG ../out/Resources/Textures/[filename].ktx
../out/Resources/Textures/[filename].png
rm ../out/Resources/Textures/[filename].ktx
```

After edited the file, convert and replace it back by doing these commands

```
compressonatorcli.exe -ff PNG -fx KTX ../out/Resources/Textures/[filename].png
../out/Resources/Textures/[filename].ktx
cd ../out/Resources/Textures/
mv [filename].ktx [filename].atc
```

Converting one into DDS

In case a texture displays into pitch black or pure white while running the game, this is how ShiVa typically behaves whenever it can't display one.
Luckily for us, this issue can be solved by converting said texture into `.dds`. This format should work universally on any platform.
For this process, you need both **PVRTexToolCLI** and **CompressonatorCLI**.

To convert a `.pvr` texture, do the following commands

```
cd ~/Downloads/out/Resources/Textures/
PVRTexToolCLI -noout -ics sRGB -i [filename].pvr -d [filename].png
rm [filename].pvr
cd ~/Downloads/compressonatorcli-4.5.52-win64/
compressonatorcli.exe -ff PNG -fx DDS ../out/Resources/Textures/[filename].png
../out/Resources/Textures/[filename].dds
rm ../out/Resources/Textures/[filename].png
```

And to convert a `.ktx` texture, do the following commands

```
cd ~/Downloads/out/Resources/Textures/
mv [filename].atc [filename].ktx
cd ~/Downloads/compressonatorcli-4.5.52-win64/
compressonatorcli.exe -ff KTX -fx DDS ../out/Resources/Textures/[filename].ktx
../out/Resources/Textures/[filename].dds
rm ../out/Resources/Textures/[filename].ktx
```

If you'd like to batch convert both `.pvr` and `.ktx` textures, you can refer back to batch converting with Bash and CompressonatorCLI's usage, then adjust the commands as you see fit.

Editing Scripts

> ***Author's Note:***
> *For this portion, you must have `Lua 5.0.3` downloaded.*
> *On Windows, grab the `lua5_0_3_Win32_bin.zip` from the luabinaries page, then unpack it.*
>
> *On Linux, you can install the `lua50` or `lua5.0` package (for `32-bit` systems) via your distro's package manager:*
> *if they can't be found, you can grab the `.zip` archive from above, unpack it, and run the `.exe`s through Wine.*
>
> *Reason why a `32-bit` variant is needed for the job:*
> *it's because ShiVa internally uses that to handle the game's logics, but also it's to mantain compatibility with older systems (**ex.** Windows XP).*
> *If you follow the instructions by using the `64-bit` variant of `Lua 5.0.3`, expect the game to softlock at any given moment.*

If you want to edit a `.lua` script, we must go inside

```
cd ~/Downloads/out/Resources/Scripts
ls -l
```

and pick whichever `.lub` script we want to edit in the first place.

Let's say i want to edit `AI_Core_Main_CarConfigurator_Function_GetAIVar.lub`:
in order to do so, we must do the following commands to decompile it

```
cp AI_Core_Main_CarConfigurator_Function_GetAIVar.lub ~/Downloads   # Do a copy of the file in
'Downloads'
cd ~/Downloads
java -jar unluac.jar -i AI_Core_Main_CarConfigurator_Function_GetAIVar.lub -o
AI_Core_Main_CarConfigurator_Function_GetAIVar.lua
```

and once edited the script, we must do the following to compile it back

```
rm *.lub    # Remove every '.lub' script in the current folder
luac50.exe -o AI_Core_Main_CarConfigurator_Function_GetAIVar.lub
AI_Core_Main_CarConfigurator_Function_GetAIVar.lua
rm AI_Core_Main_CarConfigurator_Function_GetAIVar.lua
mv -f AI_Core_Main_CarConfigurator_Function_GetAIVar.lub ~/Downloads/out/Resources/Scripts/
```

Repacking and Testing

> ***Author's Note:***
> *This process can be easily done on Windows & Linux.*
> *Reason why i'm not documenting the latter for other platforms: it's because it's more involved and convoluted to do.*
>
> *Also, the whole process will only show how to reimport a single asset.*
> *If you want to have a complete map of which folders you can reimport without breaking the game, you can safely go for:*
> *`Models`, and the following subfolders from the `Resources` folder: `AnimClips`, `Meshes`, `Musics`, `Scripts`, `Skeletons`, `Sounds`, `Textures`, and `XML`.*
>
> *If you reimport beyond those folders, even if you put the assets in the right place: expect the game to display just a black screen.*
> *Only way to solve it is picking the original `.stk`/`.smf` archive, then reimport the right folders afterwards.*

And to finish everything in beauty, we'll now learn on how to repackage our edited `.stk`/`.smf` file:
all by using **QuickBMS**, *some hacky way to make our mod work*, and **ShiVa Authoring Tool**!

The very first thing that we must do is deciding which file/s we want to reimport to our previously extracted `S3DMain.smf`.
Let's say that we have edited a texture file (as shown previously on how to edit and replace one) and we want to import it into our game and see the results.

As the official documentation[archive.org] for **QuickBMS** mentions:
if we want to make the reimporting much easier for the program, we must get rid of the unedited assets from the extracted `.smf` archive.
To make the whole thing quicker, just follow through these commands i had written below

```
cd ~/Downloads/out                  # We'll move where we had previously extracted the file from
rm -rf Games Models Scenes          # To start with, we'll remove the folders outside of the
'Resources' folder
cd Resources                        # Get into the 'Resources' folder
rm -rf `ls | grep -v Textures`      # Remove everything that isn't the 'Textures' folder
cd Textures                         # Move into the 'Textures' folder
rm `ls | grep -v DefaultFont.dds`   # Remove every file outside of the one we've edited
cd ~/Downloads/BMS/                 # In the end, move back to the QuickBMS folder
```

With those removed, now it is our time to reimport the edited assets back to the file.
If the file is *smaller* or *the same size as the original file*, write this command

```
./quickbms -w -r shiva.bms ~/Downloads/assets/S3DMain.smf ~/Downloads/out/
```

if you're wondering what's the `-w` argument for, that gives the tool permission to write on the selected archive

Else, if the edited file is ***much bigger*** than the original, write this command instead

```
./quickbms -w -r -r shiva.bms ~/Downloads/assets/S3DMain.smf ~/Downloads/out/
```

the double `-r`s, yet according to the official documentation, enables a mode called `REIMPORT2`.
With this mode enabled, `QuickBMS` will reimport any asset bigger than the original without any issues or errors to speak of.

If using the terminal ain't your strong suit, it's documentation introduces other ways that would involve using the GUI:
such as drag-n-dropping the edited files towards the `reimport` batch file, even though i'll leave the whole explanation to the docs i had linked above.

Otherwise, if you're using **AmorBMS**, you just need to do the following:

- Fill in the `Path to BMS script` path;
- Fill in the `Path to output directory` field;
- Fill in the `Path to target file` field;

After done, just click on the `Reimport` button

After the whole reimport is done, you may wonder: *what has happened to the game package?*
Into very simple words, `QuickBMS` has unintentionally broke the `.stk`/`.smf` file.

Since the tool has broke the package, ShiVa's `S3DEngine` executable has a so-called **CRC check** feature compiled inside.
This check, in a few words, compares the `CRC` offsets (which uses the `0x11112222` syntax) of the former file's and the new one's (that we had reimported previously):

- *If the file **does** pass the check*, it gets loaded into the game as-is;
- *If the file **doesn't** pass the check*, the engine won't load the "compromised" file, it'll log the issue either through the terminal or a logfile (`S3DClient.log`), and it would try to find for locally-stored replacement files inside the host machine;

Also, reason why `QuickBMS` has broke the package it's because it doesn't support `CRC` at all (which is, too, mentioned in it's docs), this causing the package to "break" when we reimport a modified asset:
though, because of this, it actually unlocked for us an *exploit* we could take advantage of.

To make our edited asset load successfully (and ignore the `CRC` check), we must copy the whole folder path whence it came from and paste it into a specific folder path the engine references from.
On Linux, we can do the following

```
cd ~/Downloads/out/     # Let's get into the path where our edited asset is at
cp -r * ~/.StoneTrip/   # And then recursively copy it's contents into the engine's folder
```

On the other hand, on Windows, you must paste the whole thing into the following folder path

```
C:\users\[username]\AppData\Roaming\Stonetrip\Player\[XXXXXXXX]\
```

if you're wondering what's `[XXXXXXXX]`, it's actually a placeholder i've put in there.
When a ShiVa-powered game is booted for the first time on Windows:
not only it will create the `Stonetrip` folder, but also one for the game aswell.

The placeholder folder should be named as an `8-digit` hexadecimal string, which is an ID that gets assigned to any brand new project the developer creates in ShiVa Engine- and after exported the game as an `.stk` archive, this ID gets embedded into it.
For simplicity's sake, let's call this ID a *project ID*.
By following this reasoning, **Dust:Offroad Racing**'s project ID is `4ee121e9`.

On Linux, the *project ID* is only useful for finding the game's savefile folder name (which can be found at `~/.StoneTrip`).

Some project IDs can either be unique or identical in some cases, but it depends from project to project.
If we were to pass an handful of `.stk` archives towards the executable that the Authoring Tool has produced earlier (this being `Dust.exe`) and then check the `C:\users\[username]\AppData\Roaming\Stonetrip\Player\` folder path afterwards:

- If there are multiple folders with each an unique project ID as it's name, this means that all of the `.stk` archives have an unique project each;
- Otherwise, if there are fewer folders and there's a specific project ID showing up often, this should be a clear sign that some of the `.stk` archives share the same project as the base (or in other words, the project has been recycled);

If you don't use Windows, dupes can also be ruled out with the following behaviors on both platforms:

- Weird behaviors will show up in-game, such as weird values showing up on-screen or unique game bugs;
- The game will overwrite the other's savefile, making all of the progress done with the previous game automatically lost;
    - To notice the latter better, launch the previous game after you've closed the new one;

This notion should come in real handy if you were to modify multiple `.stk` archives and want to rule out dupes and whatnot.

When the whole copying is done, we must now test if our mod will work:
this by recompiling the repacked `S3DMain.smf` file with **ShiVa Authoring Tool**, picking the target platform, and use the standalone `S3DEngine` executable (that'll be included inside the outputted `.zip` archive) to see if it works.
I'll save myself the whole explanation, since the process is the same as creating unofficial ports for your favorite ShiVa-powered game.

Once compiled the game, you shouldn't need to do it ever again, unless you're aiming to modify more assets from the same game.

Reason why recompilation isn't necessary is due to this:
you can actually get inside the folder path the engine refers to, and then edit the file without having to do any reimports ever again- but that's proportioned between the amount of reimported assets into the `.smf`/`.stk` archive and the locally-stored file/s inside the `Stonetrip` folder.
If you had reimported 1 asset, you can edit just 1. If it's 2, edit 2, so and so forth.

Once you had reimported one or more assets with `QuickBMS`, though:
the `.stk`/`.smf` archive will remain permanently "broken". If you re-extract the edited archive, it will only extract a partial amount of it's remaining, unedited, assets.

If you want to modify more assets, you must follow these steps:

- Move your modified asset/s *out* of the folder `QuickBMS` used to extract it's contents;
    - In our case the modified asset is at `Downloads/out/`;
- Copy the original `.smf/.stk` archive, which is `S3DMain.smf` in our case (in case some screw-ups will happen);
- Paste it in the same path as the `QuickBMS` binary is at;
- Rename it to something of different (like `Dust.stk`, for example);
- Extract it all over again with `QuickBMS` (or `AmorBMS`);
    - If you forgot how to do some of these steps, you can check back obtaining and extracting the archives;
- Convert, edit, and convert back all of the assets you want;
    - You can go back to the modding section if you forgot how to do the process;
- Reimport the edited assets with `QuickBMS` (or `AmorBMS`);
    - The last 2 steps can be switched up, if you prefer reimporting the original assets and edit them post-reimport;
- Move the edited assets into your own OS' dedicated folder path for ShiVa-powered games;
    - On Windows, move it into `C:\users\[username]\AppData\Roaming\Stonetrip\Player\4ee121e9\`;
    - On Linux, move it into `~/.StoneTrip/`;
- Repack the `.stk` archive with *ShiVa Authoring Tool*;
- Pick the target platform with the latter;
- Move and unpack the `.zip` archive that the Authoring Tool generated in the `Downloads` folder;
- Test if the mod works;

And that's it, *you've officially learned on how to create a mod for your favorite ShiVa game!*

Messing with ShiVa

> ***Author's Note:***
> *This section assumes that you had reimported the game's assets as-is with `QuickBMS`.*

After we had successfully learned on how to create mods for our favorite ShiVa-powered game and moved the assets elsewhere: from now on the assets will be completely vulnerable to any sort of editing, letting us mess with the engine in unintended ways!

From this portion onwards, though, i will teach how to do the following:

- Disabling animations;
- Swapping meshes;

Disabling Animations

> ***Author's Note:***
> *This portion assumes that you had reimported all of the assets from `/Resources/AnimClips/` with `QuickBMS`:*
> *then placed them into your OS' dedicated folder path for ShiVa-powered games firsthand.*

This portion should be pretty simple and straight-forward.
All we have to do is moving inside the folder we had placed the reimported assets at.

If you are on Windows, get into the following folder path

```
C:\users\[username]\AppData\Roaming\Stonetrip\Player\4ee121e9\Resources\AnimClips\
```

Otherwise, if you're on Linux, write this command on your terminal

```
cd ~/.StoneTrip/Resources/AnimClips/
```

After we had gotten into the folder, we will be greeted by lots of files that end up with the `.ani` extension.
If you had checked the table at more proprietary formats, these files contain all of the animation clips that the engine uses to animate *all* of the things that happen in the game.

To disable them, you just need to rename all of the files en-masse:
either via the terminal or by using a tool that could batch rename lots of files together (like **Thunar** on Linux, or Ant Renamer on Windows).

In my case, i'll easily do the latter by writing this command into the terminal

```
for file in *.ani; do mv "\$file" "\$file".bak; done
```

This will rename all of the files by adding the `.bak` extension right after their filenames.

After done, let's move back where we have the `Dust.stk` file outputted by the *Authoring Tool*:
then launch it with `S3DEngine` to see if it worked.

On Windows, open the command line and type the following commands

```
cd C:\users\[username]\Downloads\
Dust.exe Dust.stk
```

While on Linux, follow these commands

```
cd ~/Downloads
./Dust_64 Dust.stk
```

If nothing happens afterwards and it is your first time using the `Dust_64` binary:
you can simply solve the problem by changing it's mode to executable

```
chmod +x ./Dust_64
```

Afterwards, try to redo the 2 commands from above.

After launched the game, try to play it and see what has changed:
if everything moves but there are no animations playing, this means we had disabled the animations correctly.

Though, what is the use-case of disabling animations in-game?
At one side, it can be done for fun.
On the other side, though, this knowledge can be used in conjuction with tools like *NinjaRipper* and *Noesis*:
making the models easier to rig with external programs (like **Blender**, for instance) once ripped and extracted.

If you want to learn on how to use *NinjaRipper*, check out it's own usage guide:
same counts for *Noesis*, which you can check it's own usage guide aswell.

Swapping Meshes

> ***Author's Note:***
> *This portion assumes that you had reimported all of the assets from `Models/`, `/Resources/Meshes/`, `/Resources/Textures/` (must be converted into DDS in order to display them), and `/Resources/Skeletons/` with `QuickBMS`:*
> *then placed them into your OS' dedicated folder path for ShiVa-powered games firsthand.*

Swapping meshes in a ShiVa-powered game can be done for a multitude of use-cases, such as:

- Modding our game to play as a specific NPC unofficially;
  - If you feel more daring, you can also play as an object or a literal map;
- Rip unused models from the game, this by using ShiVa as a *glorified 3D model viewer*;
- Rip used models from the game without doing any progress whatsoever;

While the process mainly consists of switching files out, making this belong rather into the changing the assets portion of the **Modding** section of my docs:
i had decided to make it part of this portion, since it's a bit more involved than just switching out a couple of files around.

According to my own experiments i had done while messing up with the assets:
there are about 2 methods to swap out the meshes, even though both show the same outcome despite being slightly different procedures.

As an extra note, this whole thing can be done in conjuction with disabling animations to rip *every* model in a static pose.

First Method

This method is the easiest of the 2, which involves switching out the `.mdo` files within the `Models/` folder.

This method will allow us to take advantage of the engine and bend it to our own will by displaying any model from the assets:
even models that can only be viewed in certain conditions (**ex.** unlocking a map, paying a currency to unlock a character, etc.) or are hidden within the game's assets.

To give a quick refresh as to what `.mdo` files are, according to the table at more proprietary formats:
they are basically plain-text data that can call a multitude of assets to compose different models in-game, whether it'll be a map, object, or even a character.
Despite their own nature, these files, once swapped, can push the engine's to it's limits-
even make ShiVa crash, depending on which `.mdo` file you're switching with.

For the sake of this tutorial, let's say we want to replace our own starter car with a beefier one.
Regardless of platform, follow these steps:

- Get into `/Models/`;
  - On Windows: `C:\users\[username]\AppData\Roaming\Stonetrip\Player\4ee121e9\Models\`;
  - On Linux: `~/.StoneTrip/Models/`;
- Select `Model_Shell_Car_1.mdo`;
- Rename it by adding the `.bak` extension at the end;
- Select `Model_Shell_Car_6.mdo`;
- Copy and paste it in the same folder;
- Rename the copy into `Model_Shell_Car_1.mdo`;

With the whole switch-out done, let's move back where we have the `Dust.stk` file:
then launch it with `S3DEngine` to see if it worked

On Windows, open the command line and type the following commands

```
cd C:\users\[username]\Downloads\
Dust.exe Dust.stk
```

While on Linux, follow these commands

```
cd ~/Downloads
./Dust_64 Dust.stk
```

After opened the game and pressed the `PLAY` button:
we'll be greeted by our standard car being swapped out with the beefier one- complete with it's own mesh, sub-meshes, textures, and sounds!

This process can be rinsed and repeated even on other models aswell, as many times as you like aswell.

Second Method

Meanwhile the second method is the slightly harder one of the 2.
This one involves switching out the model's respective `.msh` and `.dds` texture in order to work.

If we were to look back at the [more proprietary formats]' table:
`.msh` files are raw mesh data, while `.dds` files are just a proprietary image format that ShiVa uses as one of it's texture formats.

This method's use-case is best used in 2 conditions:

- If the first method makes the game crash: use this method as a fallback;
- You want to make a more complicated mod, which goes beyond a simple `.mdo` file swap-out;

While the whole process may sound easy on paper, the part that makes it harder is how certain `.msh` files are stored in the assets:

- If the mesh is a simple character or object: it would be contained into one singular `.msh` file;
- If the mesh is otherwise a map or vehicle: high chances would be that both are spliced into multiple `.msh` files;

There are also exceptions to these examples, such as **Dust: Off-Road Racing** storing each of it's maps into a singular `.msh` file.

To start things in order, you *must* stay into the `Models/` folder- but why?
Remember when it was told that `.mdo` files are basically plain-text files?
Alright, we can use that knowledge to our advantage and open said file with a text editor.

If we were to use the whole "replacing our starter car with a beefier one" as the guideline for this tutorial aswell:
we must open the `Model_Shell_Car_1.mdo` file as a plain-text document.

After opened, we should be greeted by this block of text:

```
 ???NcX???? ??????????î???□???*??????? ?????ÈB??ÈB??úB???? ?□??????????????????????????????□??□?
??□??□?
???????????????\$??? ????□???Model_Shell_Car_1 ??????????f??? ????ÿ,???(???Model_Shell_Car_1_Polyg
onal_Geometry_lib□???□???ÿ□???Mat_Shell_Car_1_0_1□????????□????????
```

If we look close enough, we'll start to see a pattern in the data.
I'll translate it's file structure to make it easier for everyone:

- The first text string, `NcX`, is the file signature. You can freely ignore it;
- Afterwards, some mesh filenames will show up;

What about the *other* car we're going to swap it out with?
I'll spare the block of text, since it's file structure is pretty much the same:
except it's few instances of `Car_1` are changed into `Car_6`.

But now? It's time for file switching!
To switch out the `.msh` files, follow these steps:

- Get into `/Resources/Meshes/`;
  - On Windows: `C:\users\[username]\AppData\Roaming\Stonetrip\Player\4ee121e9\Resources\Meshes\`;
  - On Linux: `~/.StoneTrip/Resources/Meshes/`;
- Select `Model_Shell_Car_6_Polygonal_Geometry_lib.msh`;
- Rename it by adding the `.bak` extension at the end;
- Select `Model_Shell_Car_6_Polygonal_Geometry_lib.msh`;
- Copy and paste it in the same folder;
- Rename the copy into `Model_Shell_Car_1_Polygonal_Geometry_lib.msh`;

And with the mesh part done, let's go straight on switching the `.dds` textures-
though, if you had noticed from earlier, `Model_Shell_Car_1.mdo` didn't had any texture names in it's text data.

How do we find the model's associated texture name?
Let's move back into the `/Models/` folder and give a look at `Model_Core_Car_1.mdo`.
Let's read together it's first lines of data:

```
 ???NcX???? ???
?? ??? □???Particle_Nitro_Back □???Tex_Core_Sfx_Puff ,???Model_Core_Car_1_lod0_Polygonal_Geometry_
lib □???Material_Core_Car_1_0_1 □???Tex_Car_1_0_1_d_512 !
```

And at the end of the file we'll see a name: `Tex_Car_1_0_1_d_512`-
let's go into the textures folder (`/Resources/Textures/`) and check if there's *any* texture with that name.

In my case, i'll check for it by doing these commands on the terminal

```
cd ~/.StoneTrip/Resources/Textures/
ls | grep Tex_Car_1_0_1_d_512
```

And according to the output, the texture exists!

```
baem@F555U:~/.StoneTrip/Resources/Textures\$ ls | grep Tex_Car_1_0_1_d_512
Tex_Car_1_0_1_d_512.dds
```

Now it's time to switch out the textures.
Just follow these steps:

- Get into `/Resources/Textures/`;
  - On Windows: `C:\users\[username]\AppData\Roaming\Stonetrip\Player\4ee121e9\Resources\Textures\`;
  - On Linux: `~/.StoneTrip/Resources/Textures/`;
- Select `Tex_Car_1_0_1_d_512.dds`;
- Rename it by adding the `.bak` extension at the end;
- Delete the original file;
- Select `Tex_Car_6_0_1_d_512.dds`;
- Copy and paste it in the same folder;
- Rename the copy into `Tex_Car_1_0_1_d_512.dds`

Before heading on to open `S3DEngine` to see if our modifications have worked:
if your game has a `/Resources/Skeletons/` folder, switching it out between a `.ske` file and another is pretty much the same process as the latter.
The secret all lies within opening it's own dedicated `.mdo` file (which you can find at `/Models/`) and open it with a text editor.
Typically the skeleton's filename should be after the mesh's or texture's filename- just keep your eyes peeled for patterns in the data!

If you skip this crucial step, though:
expect the model to render on-screen as an eldritch horror made out of polygons.

After we had switched out *both* meshes and textures, let's move back where `Dust.stk` should be usually at:
then launch it with `S3DEngine` to see if it worked

On Windows, open the command line and type the following commands

```
cd C:\users\[username]\Downloads\
Dust.exe Dust.stk
```

While on Linux, follow these commands

```
cd ~/Downloads
./Dust_64 Dust.stk
```

Once the game has opened, the game's menu has shown up, and clicked on the `PLAY` button:
here it is, our modification in all of it's glory!

Distributing your Mod

Now that you've saw on how to *make* a mod, how can we *distribute* it?

It's actually simple, we just need to take the `.zip` we have obtained from the *Authoring Tool* and add inside every file coming from the `Stonetrip/Resources/` folder path!

If you're on Linux, do the following

```
cd /media/[username]/USB                        # Let's get into the USB to find the
'Dust_Linux.zip' archive
7z a Dust_Linux.zip ~/.StoneTrip/Resources    # Use 7zip to include the whole contents of
'Stonetrip/Resources/' inside of it
```

If you're Windows, do these steps:

- Get inside of your USB via your file explorer;
- Find the `Dust_Windows.zip` archive;
- Right click on it and click on `7-Zip>Add to "Dust_Windows.zip"`;
- When `7-zip` opens, copy `C:\users\[username]\AppData\Roaming\Stonetrip\Player\4ee121e9\` and paste it inside the path field;
  - Make sure to replace `[username]` with your own username;
- Select the `Resources` folder;
- Click on the `Add` button;
- Let the program do it's magic!

If you feel eager to add a `readme` file for instructions on how to play and install the mod:
you can add it to the archive by repeating the whole process above.

And when the whole thing is done, *you can now distribute your mod wherever you like!*

Epilogue

And here we are, right at the end of the documentation.
I hope all of the information was as exhaustive as possible regarding this dead, yet nichè, engine that is **ShiVa Engine**.
Didn't expect how it relied to a great chunk of FOSS tools in the first place, yet i'm enough glad that the developers relied on such and didn't hide any information to such in the first place.

Hoping that reading throughout all of this was fun and entertaining as much as the time (and years) i've took between tinkering and experimenting with ShiVa to figure it's in-and-outs (but also must give credit on all of the self-improvement i've took in the first place).

And now?
All of this information isn't hidden anymore to the internet. You're totally welcome!

Sources

- Wayback Machine, for being incredibly useful for both my research on the company & the engine itself, but also thanks to the users that archived the tools i had referenced into the documentation;
- XenTax and ZenHax, had very relevant info on the engine itself, such as `.mdo` files containing plain text data, and `.smf` and `.stk` files being proprietary archives;
- DLL Files.com, used it as a reference for any `.dll` file contained within the engine;
- Online TrID by *Marco Pontello*, which it introduced me years ago to the world of *file signatures*, but also used it on my journey for figuring out certain proprietary formats along the way;
- OpenJDK.org & Microsoft Learn, builds and documentation for `OpenJDK`;
- i3DConverter, the *only* tool on the internet that gave me the hint that `.msh` files contain actual raw model data;
- PowerVR Developer Documentation, documentation for *PVRTexTool* and the `.pvr` file format;
- 3D Game Development with ShiVa3D Suite: AI & Animation, documentation on how to use `.ste` and `.stk` archives;
- Android Developers, documentation on how to enable *Developer Options* and *USB Debugging* on Android;
- Lua.org, manpage for the `5.0` edition of the `luac` compiler;
- Sourceforge, source for unluac and Lua Binaries;
- QuickBMS, documentation for *QuickBMS*;
- GitHub, source for ShiVa Technologies' page, QuickBMS' mirror repository, and Compressonator;
- Archive Today, secondary source for archived webpages;
- Khronos Registry, technical documentation for the `.ktx` file format;
- MCV/Develop, linked some of their articles in the *Quick History* section regarding **StoneTrip**/**ShiVa SAS**;
- **Myself**, dedicated years of my life to figure out the engine's ins-and-outs, created *AmorBMS*, and archived some stuff for the docs;